

Mariusz Wieczorek

mariusz.wieczorek@kabat.pl

Streszczenie

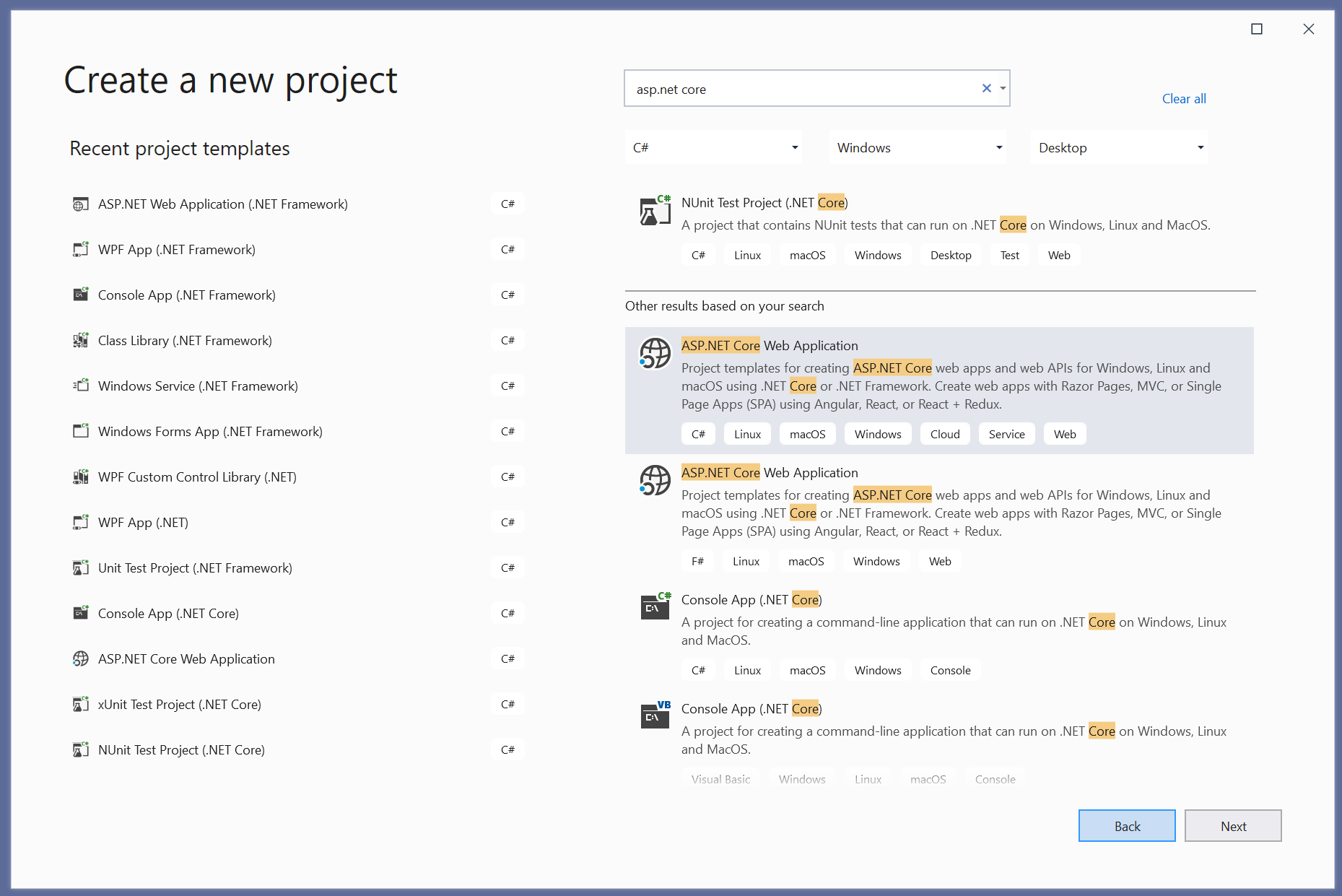
Tworzymy aplikacje webowej z wykorzystaniem szablonu .Net Core MVC  
Omówienie powstałej struktury plików  
Co to jest Areas  
Klasa Startup.cs i metody Configure i ConfigureServices,  
Klasa Program.cs – punkt startowy aplikacji

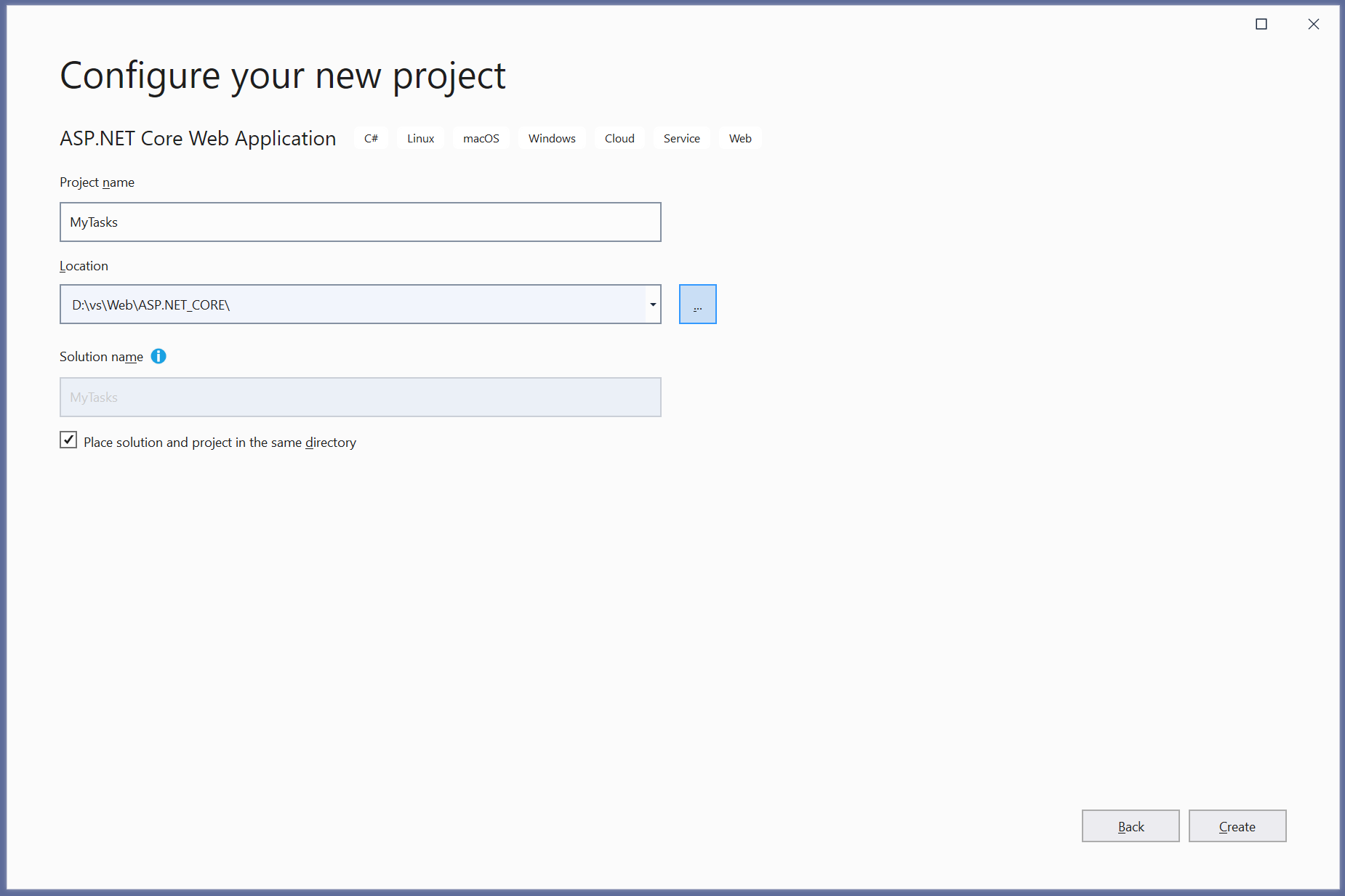
asp.net Core

Utworzenie aplikacji z szablonu

**Tworzymy Aplikację ASP.NET Core**

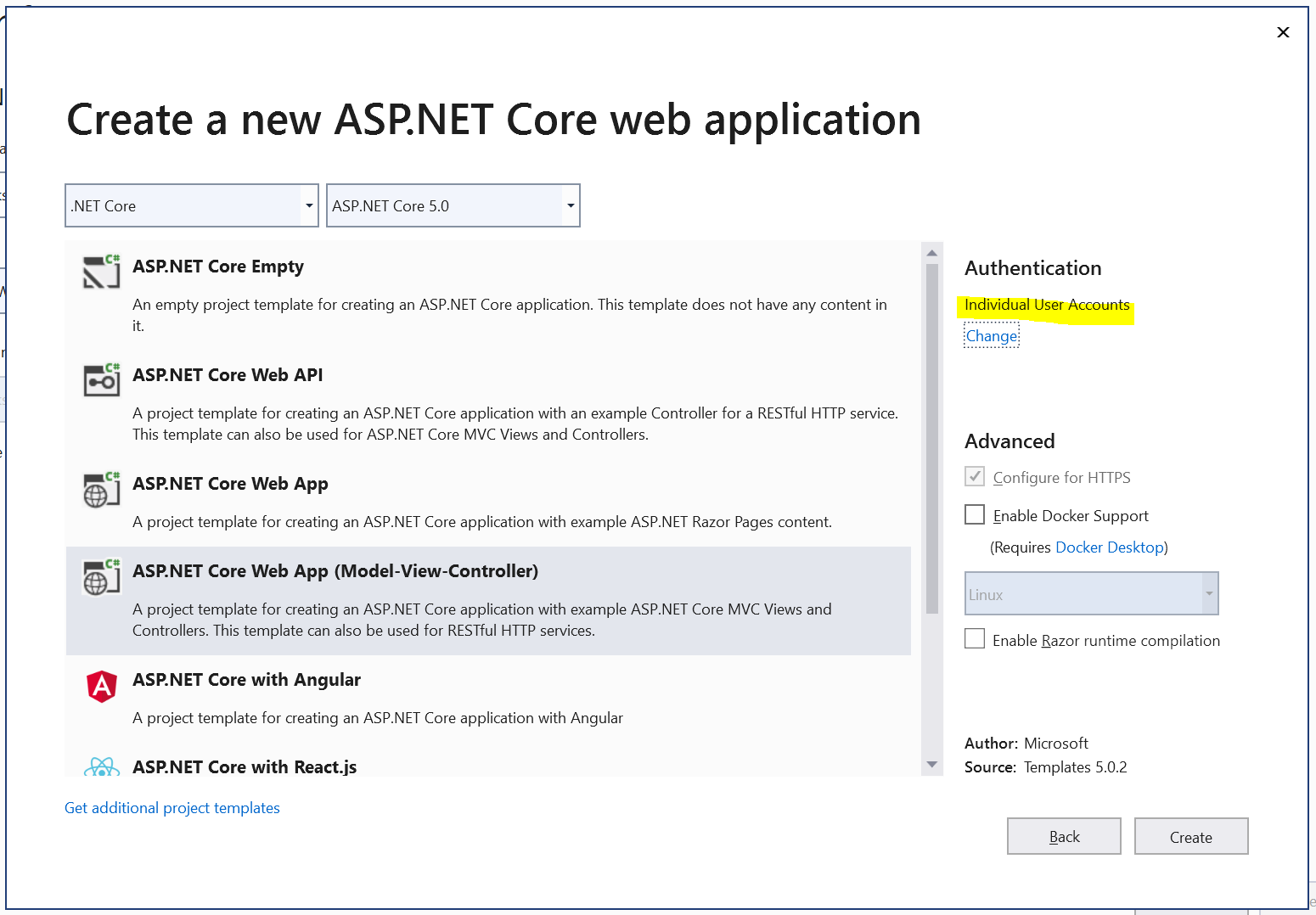
Tworzymy nowy projekt, w polu wyszukiwania wpisujemy asp.net core, wybieramy wersję z c#





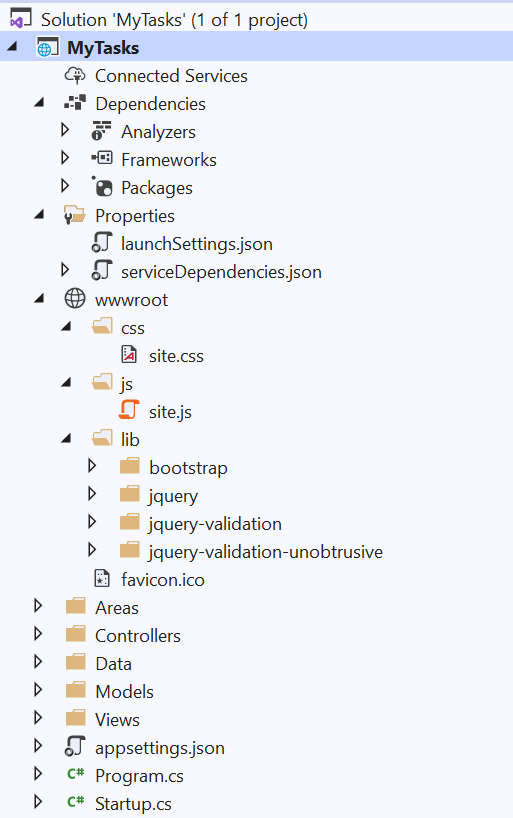
Na tym ekranie możemy wybrać aplikację Web API, możemy wybrać aplikację webową zgodną z MVC lub też trochę inną odmianę aplikacji webowej **Razor Pages**. Może również zostać utworzony projekt z szablonem Angulara, Reacta itd.

My ustawiamy aplikację webową **ASP.NET Core Web App** ze wzorcem **MVC**  
Wybieramy wersję .NET Core, dodatkowo konfigurujemy autentykację, aby wygenerował się odpowiedni szablon do zarządzania kontami użytkowników.



Klikamy Create.

Powstała struktura jest bardzo podobna do zwykłego ASP.NET MVC, jest jednak kilka zmian.

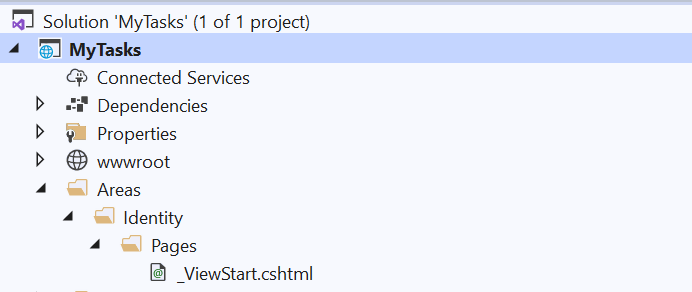
  
  
W **Dependencies** – referencje do bibliotek zainstalowanych w projekcie

W **Properties** – dwa pliki – pierwszy z informacjami uruchomieniowymi, a drugi z opisanymi zależnościami.

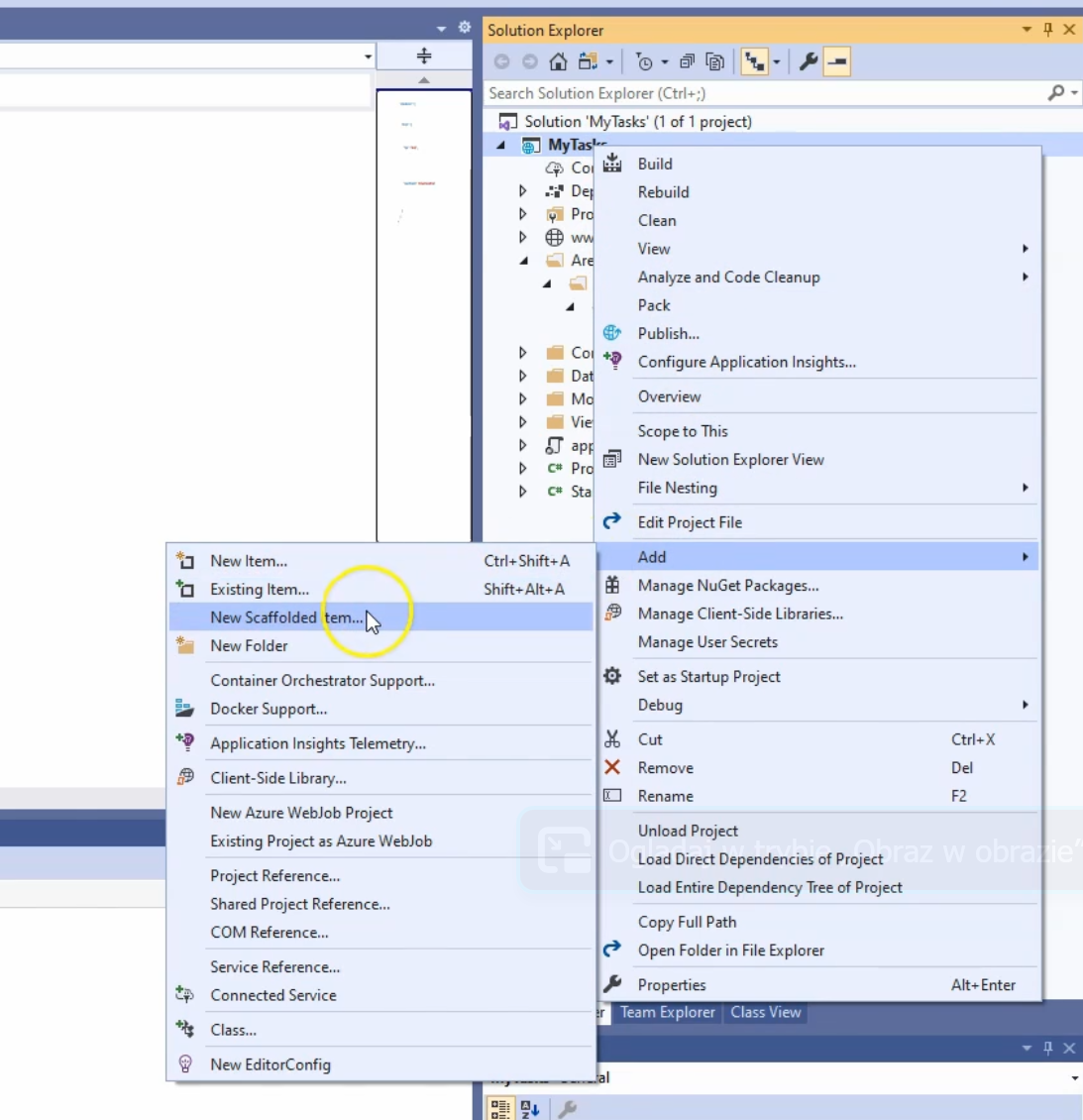
W [www.root](http://www.root) są tu zarówno skrypty **javascript**, pliki **css** jak i biblioteki javascript zainstalowane w projekcie, oraz favicon.ico

**Areas** – obszary, strefy. Jeśli nasza aplikacja się bardzo rozrasta, to możemy ją podzielić na moduły.  
Np. system księgowość ma moduł faktury, użytkownicy, podatek itd. Możemy wydzielić faktury do jednej strefy np. Invoice. Będą tu katalogi zawierające modele widoki i kontrolery.

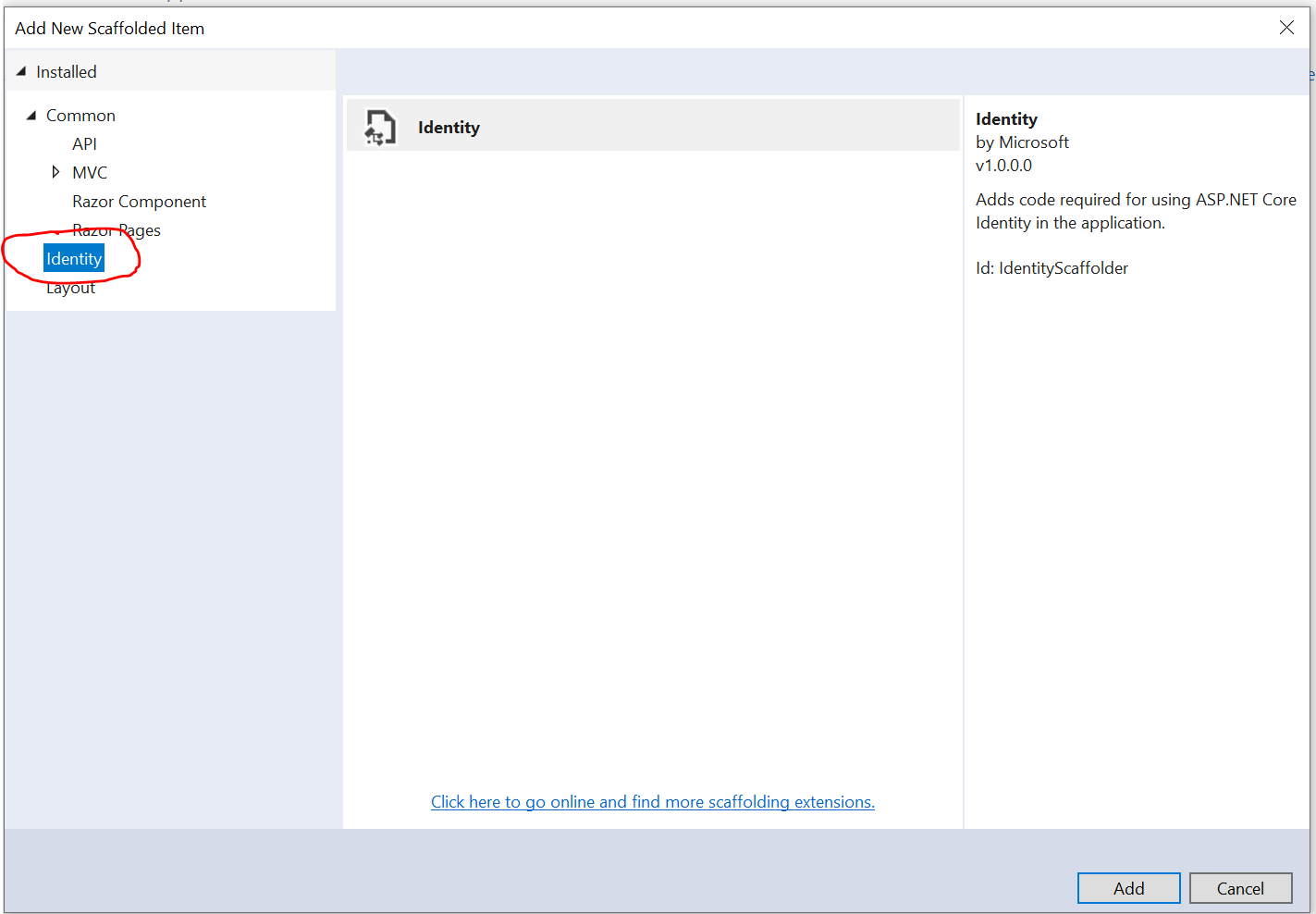
Nasz projekt zawiera już zarządzanie użytkownikami, ale nigdzie nie widać kodu za to odpowiedzialnego. Został on właśnie wydzielony do osobnej strefy **Identity**



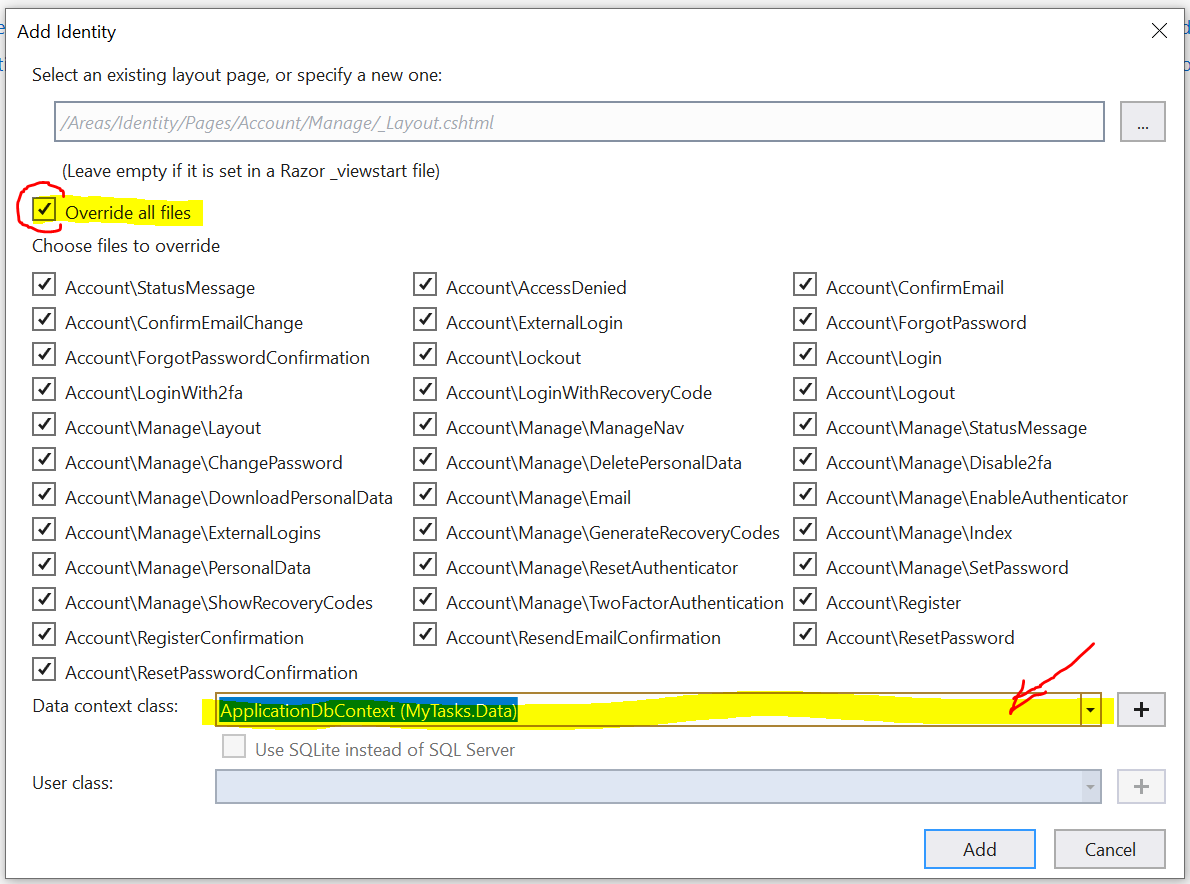
Możemy tę strefę przywrócić do naszego projektu – klikamy prawym przyciskiem na projekt, następnie wybieramy **Add** i **New Scaffolded Item**

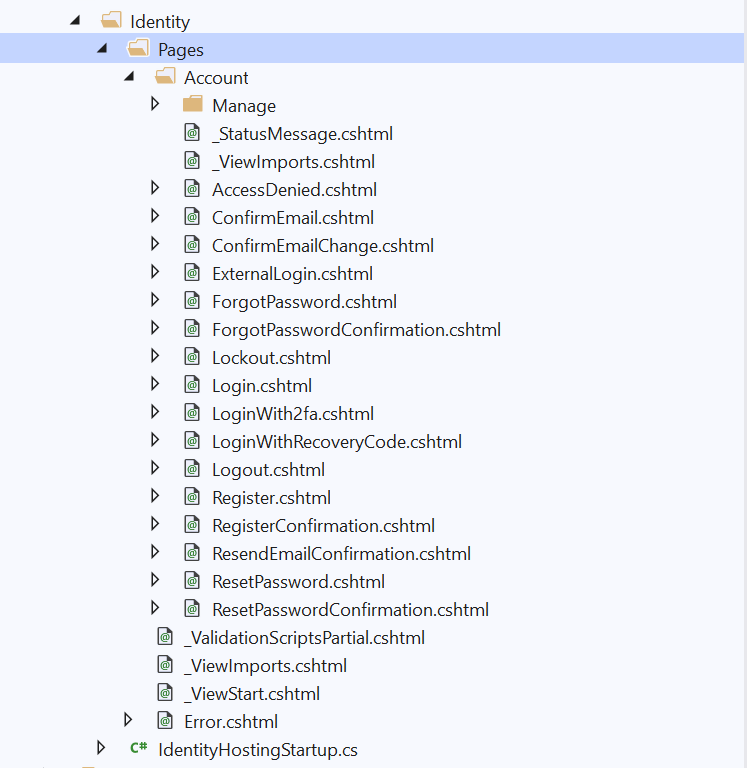


Wybieramy Identity, następnie Add

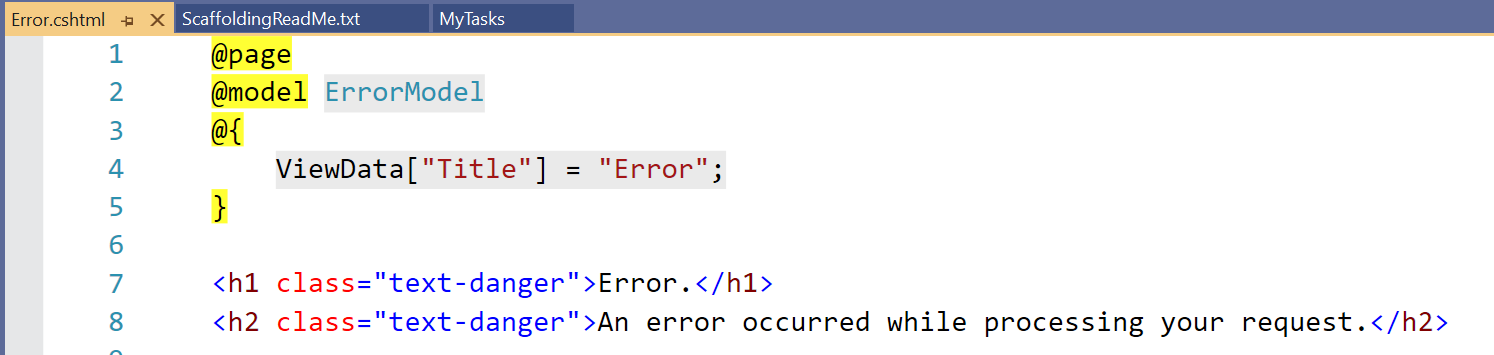


Możemy wybrać wszystkie pliki, które chcemy dodać do naszego projektu, na górze osobny layout,  
dla tego obszaru , następnie wybieramy datacontext



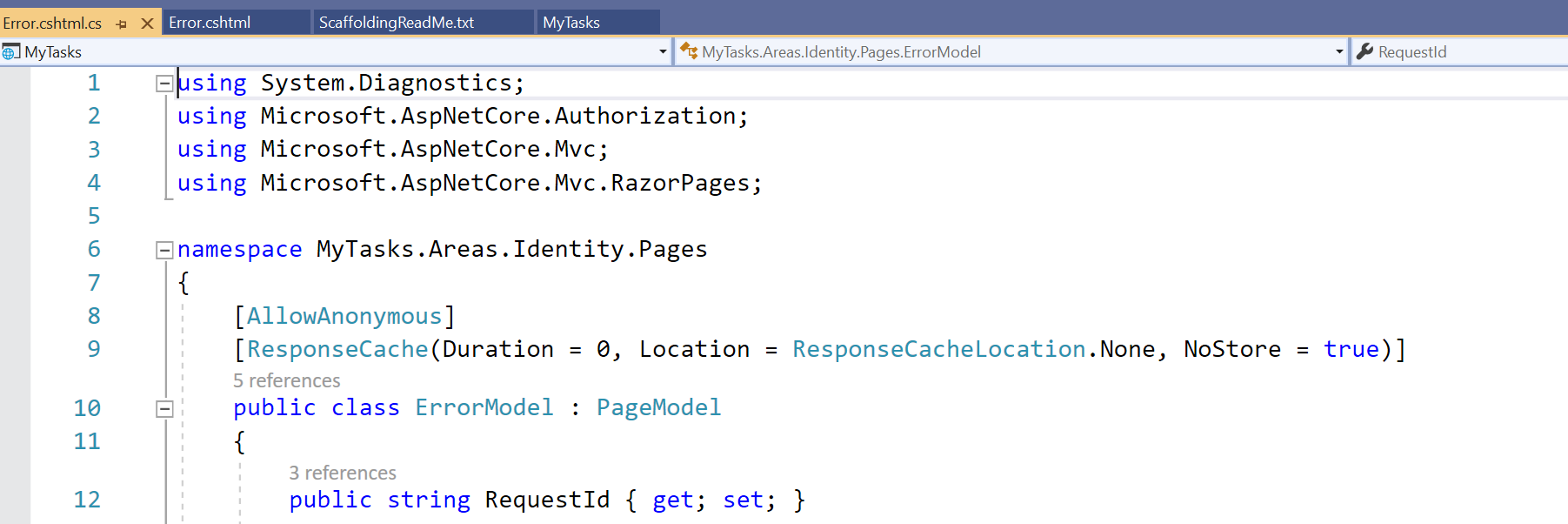
Został dodany cały obszar identity w folderze Areas. Pojawi się mnóstwo nowych plików  


Pliki te są trochę inne niż w MVC, są napisane w **Razor Pages** wprowadzony w ASP.NET 2.0  
Składnia przypomina MVC, ale jest trochę inna.  
**Widoki Razor** rozpoczynają się od @page, sprawia to, że plik jest akcją MVC i obsługuje żądania bezpośrednio, bez przechodzenia przez kontroler.



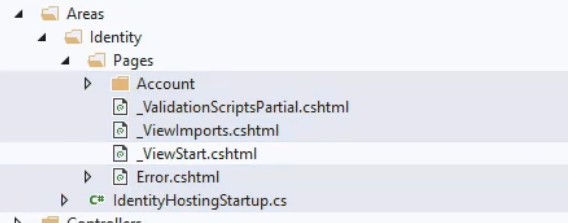
Pliki w folderze pages dzielą się na dwa, pierwszy to strona Razora z rozszerzeniem .cshtml, oraz drugi z rozszerzeniem .cs i tu mamy c#



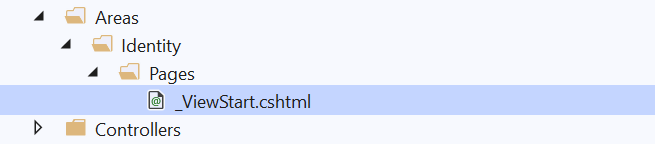


Potem dalej standardowo modele, widoki

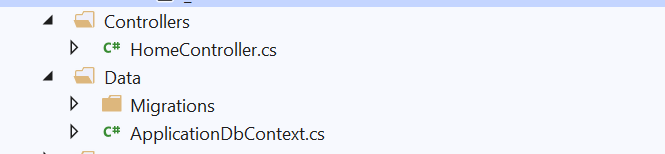
Nie będziemy tego kodu potrzebować, usuwamy wszystko za wyjątkiem \_ViewStart.cshtml



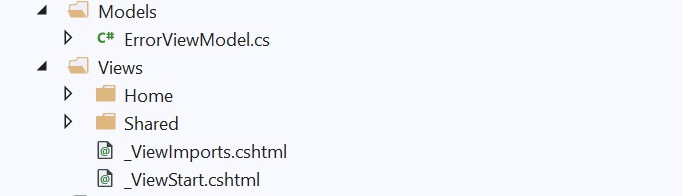
Tak, że zostaje tylko ViewStart



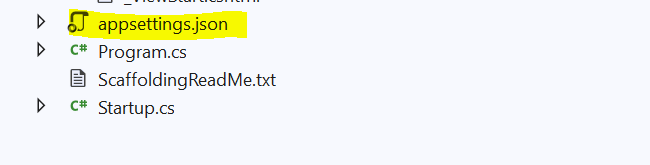
Dzięki temu nasz Layout będzie wspólny we wszystkich stronach naszej aplikacji.  
Jeżeli w przyszłości chciałbyś zrobić jakieś zmiany w zarządzaniu użytkownikami, to musimy zostawić ten cały katalog.

Następnie mamy standardowo folder dla kontrolerów, katalog Data gdzie mamy pliki z migracjami oraz ApplicationDbContext.cs z entity framework core.  


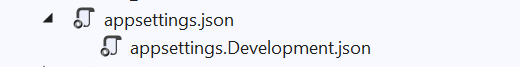
Dalej standardowo Modele i Widoki, bez zmian



Następnie **zamiast pliku Web.config** w formacie xml’a mamy **appsettings.json**



Plik ten jest już na wstępie podzielony na dwa pliki, jeden produkcyjny, drugi rozwojowy



Jeden produkcyjny, drugi rozwojowy/

Oraz Program.cs i metoda Main(), od której startuje nasza aplikacja.  
Konfiguracje serwer www, główny folder treści, integracje z IIS, definiuje klasę startup  
oraz buduje i uruchamia skonfigurowany obiekt webhost

|  |
| --- |
| public class Program  {  public static void Main(string[] args)  {  CreateHostBuilder(args).Build().Run();  }  public static IHostBuilder CreateHostBuilder(string[] args) =>  Host.CreateDefaultBuilder(args)  .ConfigureWebHostDefaults(webBuilder =>  {  webBuilder.UseStartup<Startup>();  });  } |

**Startup.cs**

Na koniec Startup.cs, który służy do skonfigurowania i uruchomienia aplikacji.

|  |
| --- |
| public class Startup  {  public Startup(IConfiguration configuration)  {  Configuration = configuration;  }  public IConfiguration Configuration { get; }  // This method gets called by the runtime. Use this method to add services to the container.  public void ConfigureServices(IServiceCollection services)  {  services.AddDbContext<ApplicationDbContext>(options =>  options.UseSqlServer(  Configuration.GetConnectionString("DefaultConnection")));  services.AddDatabaseDeveloperPageExceptionFilter();  services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = true)  .AddEntityFrameworkStores<ApplicationDbContext>();  services.AddControllersWithViews();  }  // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.  public void Configure(IApplicationBuilder app, IWebHostEnvironment env)  {  if (env.IsDevelopment())  {  app.UseDeveloperExceptionPage();  app.UseMigrationsEndPoint();  }  else  {  app.UseExceptionHandler("/Home/Error");  // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.  app.UseHsts();  }  app.UseHttpsRedirection();  app.UseStaticFiles();  app.UseRouting();  app.UseAuthentication();  app.UseAuthorization();  app.UseEndpoints(endpoints =>  {  endpoints.MapControllerRoute(  name: "default",  pattern: "{controller=Home}/{action=Index}/{id?}");  endpoints.MapRazorPages();  });  }  } |

W projekcie mamy wbudowany **kontener dependency injection**, możemy go skonfigurować w metodzie ConfigureServices()

W Metodzie Configure() możliwość skonfigurowania potoku http,  
reakcja na wyjątki aplikacji w zależności czy to środowisko produkcyjne czy deweloperskie,  
udostępnianie statycznych plików w wwwroot, domyślny routing.